Tipuri de date

Limbajul Java suportă următoarele tipuri primitive de date:

- numerice:
    - întregi: byte (1 octet), short (2), int (4), long (8);
    - reale: float (4 octeţi), double (8)
- caracter: char (2 octeţi);
- logic: boolean (true, false)

Tip de date	**Valoare minimă**	**Valoare maximă**
byte	-128	127
short	-32768	32767
int	-2147483648	2147483647
long	-9223372036854775808	9223372036854775807
float	1.4E-45	3.4028235E38
double	4.9E-324	1.7976931348623157E308

Programul care furnizează valorile din tabelul anterior este următorul:

```
Byte bit=new Byte("8");
System.out.println("Minim byte = " + bit.MIN_VALUE);
System.out.println("Maxim byte = " + bit.MAX_VALUE);

Short scurt=new Short("0");
System.out.println("Minim short = " + scurt.MIN_VALUE);
System.out.println("Maxim short = " + scurt.MAX_VALUE);

Integer intreg=new Integer(0);
System.out.println("Minim int = " + intreg.MIN_VALUE);
System.out.println("Maxim int = " + intreg.MAX_VALUE);
```

```
Long lung = new Long(0);
System.out.println("Minim long = " + lung.MIN_VALUE);
System.out.println("Maxim long = " + lung.MAX_VALUE);

Float flotant = new Float(0);
System.out.println("Minim float = " + flotant.MIN_VALUE);
System.out.println("Maxim float = " + flotant.MAX_VALUE);

Double dublu = new Double(0);
System.out.println("Minim dublu = " + dublu.MIN_VALUE);
System.out.println("Maxim dublu = " + dublu.MAX_VALUE);
```

Java este un limbaj complet orientat obiect şi ca urmare, în afară de tipurile primitive de date, restul sunt tipuri referinţă. Un obiect ocupă o anumită zonă de memorie de la o anumită adresă, iar manipularea lui se realizează prin intermediul referinţei (adresei) la acel obiect.

Variabile şi constante

În lucrul cu variabilele şi constantele apar două etape foarte importante: declararea (prin care se specifică numele şi tipul) şi iniţializarea (prin care se atribuie o valoare efectivă).

Declararea variabilelor se realizează prin specificare tipului şi a numelui, astfel:

tip Nume_variabilă;

Exemplu:

```
int a;          //variabile a este de tip int
String sir;     //variabila sir este de tip String
```

Iniţializarea variabilelor presupune operaţiunea de atribuire a unei valori:

```
a = 20;
sir = ”abcde”;
```

Constantele se caracterizează prin faptul că odată ce au primit o valoare, nu se mai pot modifica. Declararea constantelor se face astfel:

final tip Nume_constantă:

Exemplu:

```
final int a;
a = 20;
```

După iniţializarea constantei a cu valoarea 20, orice încercare de a schimba această valoare va eşua.

În Java, operaţiunile de declarare şi iniţializare pot fi reunite într-o singură linie, în maniera următoare:

```
public static void main(String[] sir) {
        int varsta = 20;     //aici se realizează declararea si iniţializarea variabilei varsta
    String nume = "Ionescu";
      System.out.println(nume + " are " +  varsta + " ani");
   }
```

Ca rezultat, pe consolă va fi afişat mesajul:

```
Ionescu are 20 ani
```

Dacă variabilele sunt doar declarate, fără a fi iniţializate, vor genera o eroare în momentul în care se va încerca folosirea lor în cadrul programului:

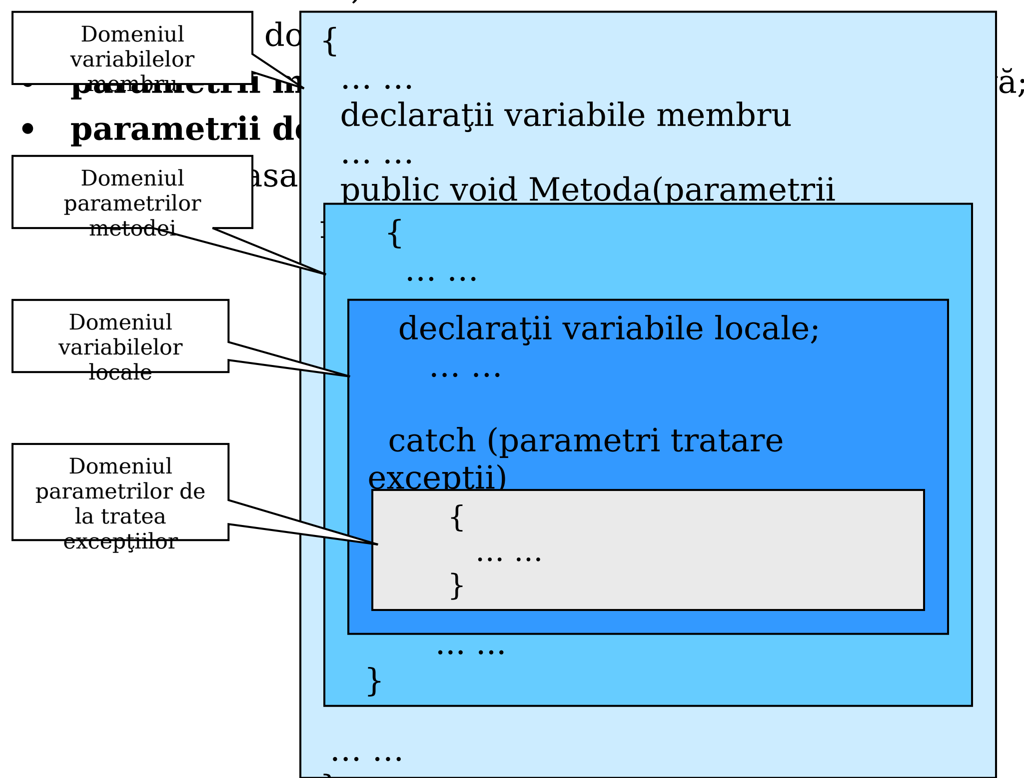
```
public static void main(String[] sir) {
        int varsta;      //aici doar se declară variabila varsta, fără a se iniţializa
      String nume;
      System.out.println(nume + " are " +  varsta + " ani");
   }
```

Ca rezultat, utilizatorul va primi următoarele mesaje de eroare:

```
variable nume might not have been initialized
variable varsta might not have been initialized
```

În funcţie de locul în care sunt declarate, variabilele se împart în următoarele categorii:

- **variabile membre**, declarate în interiorul unei clase, vizibile pentru toate metodele clasei respective şi pentru alte clase în funcţie de nivelul lor de acces (se va reveni ulterior la acest aspect);
- **variabile locale**, declarate într-o metodă sau într-un bloc de do…
- **parametrii m**… ă;
- **parametrii d**… …asa…



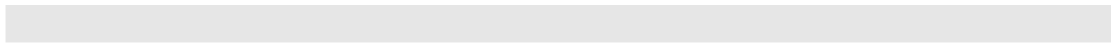


Figura nr. 1 – Domeniul de vizibilitate pentru variabile în Java

Un nume de variabilă este valid dacă nu este un cuvânt cheie (un cuvânt Java) şi dacă este unic în cadrul domeniului să de vizibilitate. Se recomandă ca denumirile de variabile să înceapă printr-o literă mică, iar denumirile de clase printr-o literă mare.

Lucrul cu vectori în Java

Un vector (sau tablou) reprezintă o structură de date care are rolul de a permite stocarea datelor de acelaşi tip. Procesul de folosire a vectorilor presupune parcurgerea a 3 paşi:

- **declararea**:
  tip[ ] Nume_vector; sau
  tip Nume_vector[ ];

Exemplu: int salariu[];
String nume[];

- **instanţierea**: se realizează prin operatorul **new** şi are rolul de a aloca vectorului memorie suficientă, în funcţie de numărul de elemente specificat.
  Nume_vector = **new** tip[dimensiune];

Exemplu: salariu = new int[100]; *//se alocă memorie pentru 100 de întregi*

Nume = new String[100]; *//se alocă memorie pentru 100 de elemente de tip String*

Un vector poate fi declarat şi instanţiat simultan:

tip[] Nume_vector = new tip[dimensiune];

- **iniţializarea**: această etapă este opţională şi se foloseşte doar în cazurile în care se doreşte ca odată cu declararea vectorului, acesta să ia şi valori efective. În acest caz lipseşte etapa de

instanţiere care se realizează automat în funcţie de numărul de elemente cu care se iniţializează vectorul.

Exemplu: String nume[] = { ”Ionescu”,”Popescu”,”Vasilescu” }

Elementele vectorului sunt accesate prin indice; este important de reţinut că primul element din vector are indicele 0, şi astfel ultimul element are indicele n-1, unde n reprezintă numărul de elemente din vector.

Deseori se cade în capcana încercării de a stabili mărimea vectorului direct, fără a folosi operatorul new; acest lucru nu este permis în Java.

```
int salariu[20];                    //incorect
int salariu[ ] = new int[20];       //corect
```

Dimensiunea unui vector se poate afla cu ajutorul cuvântului cheie **length**.

```
int salariu[ ] = new int[20];
System.out.println(”Dimensiunea vectorului salariu este ” + salariu.length);     //20
```

În Java se poate lucra foarte uşor cu tablouri multidimensionale deoarece ele pot fi considerate ca fiind vectori de vectori.

```
int salariu[ ] = new int[5][20];          //matrice cu 5 linii şi 20 de coloane (total 100 elemente)
```

Astfel, salariu[0][0] este primul element din matrice, iar salariu[4][19] este ultimul element din matrice.

Copierea vectorilor se face prin apelarea metodei:

System.arraycopy*(vector_sursă, poz_vector_sursă, vector_dest, poz_vect_dest, nr_elemente)*

unde:

- *vector_sursă* reprezintă vectorul ce va fi copiat;
- *vector_dest* este vectorul în care se vor copia elementele;
- *poz_vector_sursă* reprezintă poziţia din vectorul sursă de la care se face copierea;
- *poz_vector_dest* este poziţia din vectorul destinaţie începând cu care se face copierea;
- *nr_elemente* reprezintă numărul de elemente ce vor fi copiate.

Astfel, pentru a copia un vector în altul se poate apela metoda System.arraycopy cu următorii parametri:

```
String nume[ ] = { ”Ionescu”, ”Popescu”, ”Vasilescu” };
String alte_nume[ ] = new String(3);
System.arraycopy(nume, 0, alte_nume, 0, nume.length);
```

1. Operatori şi structuri de control în Java

Ca şi o fiinţă sensibilă, un program trebuie să manipuleze lumea în care se desfăşoară şi să facă alegeri în timpul execuţiei.[1]

În Java, obiectele şi datele sunt manipulate prin intermediul operatorilor, iar alegerile se fac prin structuri de control specifice. Java are ca bază de plecare limbajul C++, şi ca urmare majoritatea operatorilor şi a structurilor de control sunt asemănătoare cu cele din C++.

1.1. Operatori Java

În principiu, un operator primeşte unul sau mai multe argumente şi produce o nouă valoare. În Java, aproape toţi operatorii lucrează doar cu primitive, excepţiile fiind ’=’, ’= =’ şi ’!=’ care funcţionează cu toate obiectele. În plus, clasa **String** suportă ’+’ şi ’+=’.

Precedenţa operatorilor este asemănătoare cu cea din alte limbaje de programare: într-o expresie se execută mai întâi operaţiile de ordinul I (înmulţirea şi împărţirea) şi apoi operaţiile de ordinul II (adunarea şi scăderea). Programatorul poate modifica precedenţa operatorilor prin folosirea parantezelor rotunde.

Principalii operatori Java sunt următorii:

- **atribuirea:** =

Operatorul de atribuire permite asignarea valorii membrului din partea dreaptă membrului stâng. Membrul drept poate fi o constantă, o variabilă sau o expresie care produce o valoare, iar membrul stâng trebuie să fie obligatoriu o variabilă. De exemplu următoarea secvenţă:

A = 4;
B = 5;
A = B;

va face ca în final variabilele A şi B să aibă ambele valoarea 5.

În cazul în care atribuirea se foloseşte în cazul obiectelor, lucrurile stau puţin diferit datorită faptului că atunci când manipulăm un obiect, ceea ce manipulăm este de fapt referinţa la

[1] Bruce Eckel, *“Thinking in Java”*, 2nd Edition, Pretince Hall, New Jersey

acel obiect, şi ca urmare atunci când atribuim „un obiect altui obiect” operaţiunea care se efectuează constă de fapt în copierea referinţei dintr-un loc în altul. Aceasta înseamnă că dacă de exemplu atribuim **C = D** în cazul obiectelor, ceea ce vom obţine este faptul că după această atribuire atât C cât şi D fac referire la acelaşi obiect la care iniţial făcea referire doar D. Ca urmare, prin operaţiunea de atribuire în cazul obiectelor nu se obţine încă o copie fidelă, ca în cazul variabilelor simple.

Exemplul următor demonstrează diferenţa care există între obiecte şi variabile simple în cazul atribuirii:

```
class Numar {
  int valoare;
}

public class atribuire {
  public static void main(String[] args) {
    int A, B;
    Numar n1 = new Numar();    //se creeaza 2 obiecte din clasa Numar
    Numar n2 = new Numar();
    A = 30;
    B = 40;
    System.out.println("A=" + A + "  B=" + B);
    A = B;                        //atribuirea în cazul variabilelor
    System.out.println("dupa A=B se obtine A=" + A + "  B=" + B);
    A = 50;
    System.out.println("dupa A=50 se obtine A=" + A + "  B=" + B);
    n1.valoare = 5;
    n2.valoare = 7;
    System.out.println("n1=" + n1.valoare + " n2=" + n2.valoare);
    n1 = n2;                      //aici se realizeaza atribuirea referinţei obiectului
      System.out.println("dupa n1 = n2 se obtine n1= " + n1.valoare + "
n2=" + n2.valoare);
    n1.valoare = 10;
     System.out.println("dupa n1.valoare=10: se obtine n1= " + n1.valoare
+ " n2=" +
          n2.valoare);
  }
}
```

Ieşirea produsă este următoarea:

```
A=30  B=40
dupa A=B se obtine A=40  B=40
dupa A=50 se obtine A=50  B=40
n1=5 n2=7
dupa n1 = n2 se obtine n1= 7 n2=7
dupa n1.valoare=10: se obtine n1= 10 n2=10
```

Exemplul nr. 2 – Atribuirea în cazul variabilelor şi al obiectelor

Se observă că în cazul obiectelor, ”modificarea” lui **n1** produce şi modificarea lui **n2**. Acest lucru se datorează faptului că n1 şi n2 nu sunt obiecte, ci **referinţe** la acelaşi obiect după operaţiunea de atribuire n1 = n2.

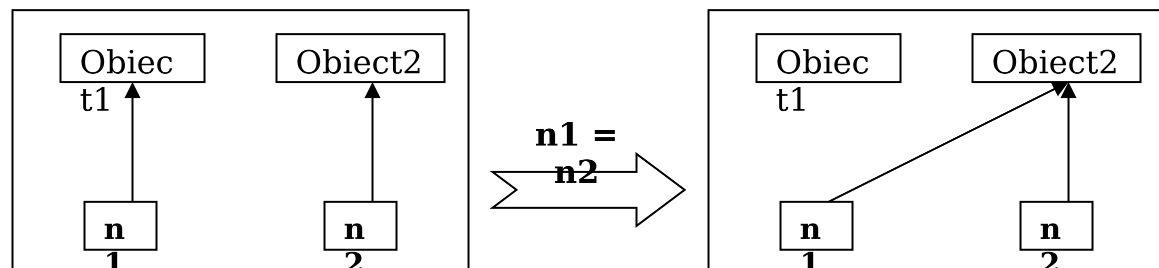


Figura nr. 2 – Operaţiunea de atribuire în cazul obiectelor

Acest fenomen este cunoscut sub numele de *”aliasing”* şi este modul de bază în care se lucrează cu obeiecte în Java; în cazul în care de exemplu nu dorim să se realizeze atribuirea referinţei aşa cum este prezentată în figură, putem folosi varianta:

n1.valoare = n2.valoare

În acest din urmă caz, n1 şi n2 reprezintă în continuare referinţe la obiecte distincte, iar atributul valoare al obiectului referit prin n1 devine egal cu atributul similar al obiectului referit de n2.

- **operatori matematici:** +, -, *, /, %

Java implementează operatorii matematici comuni pentru majoritatea limbajelor de programare: adunarea (+), scăderea (-), înmulţirea (*), împărţirea(/) şi modulo (%, care furnizează restul împărţirii între doi întregi). La fel ca în C++, în Java există o notaţie prescurtată de forma lval op=rval.

De exemplu, pentru a adăuga 7 la A se poate folosi expresia **A+=7**, care este echivalentă cu **A=A+7**.

Pentru autoincrementare şi autodecrementare există operatori specifici în două variante: post şi pre.

Operatorul de autoincrementare **++** are ca efect ”creşterea cu unu” a valorii variabilei respective; dacă **A** este de tip întreg, expresia **++A** este echivalentă cu **A=A+1**. În mod analog se comportă şi operatorul de autodecrementare (--).

În cazul post-incrementării (A++) şi a post-decrementării (A--), valoarea este produsă şi apoi se realizează operaţiunea; în

cazul pre-incrementării (++A) şi a pre-decrementării (--A), mai întâi se realizează operaţiunea şi apoi se produce valoarea obţinută.

```
public class incrementare {
  public static void main(String[] arg) {
    int i=1;
    System.out.println("i: " + i);
    System.out.println("++i: " + ++i);  //pre-incrementare
    System.out.println("i++: " + i++);  //post-incrementare
    System.out.println("i: " + i);
    System.out.println("--i: " + --i);  //pre-decrementare
    System.out.println("i--: " + i--);  //post-decrementare
    System.out.println("i: " + i);
  }
}
```

Ieşirea produsă este următoarea:

```
i:      1
++i:    2       //i se incrementează şi apoi se afişează
i++:    2       //întâi se afişează i şi apoi se incrementează
i:      3
--i:    2
i--:    2
i:      1
```

Exemplul nr. 3 – Pre/Post-decrementare, Pre/Post-incrementare

Se poate observa că în cazul formei prefixate obţinem valoarea după ce s-a efectuat operaţiunea, în timp ce în cazul formei postfixate valoarea o obţinem înainte de efectuarea operaţiunii.

- **operatori logici:** &&(AND), ||(OR), !(NOT)

Operatorii logici produc o valoare booleană (*true* sau *false*) în funcţie de argumentele care-i însoţesc. În Java, operatorii logici se pot aplica doar valorilor de tip boolean.

O secvenţă de genul:

```
int a=20;
int b=30;
if ( (a>10) && (b>25) )
System.out.println(”A şi B sunt mai mari decât limitele”);
else
        System.out.println(”A şi B nu sunt mai mari decât limitele”);
```

va avea ca rezultat afişarea ”A şi B sunt mai mari decât limitele”.

- **operatori relaţionali:** <, <=, >, >=, = =, !=

Operatorii relaţionali generează un rezultat de tip boolean; o expresie relaţională produce *true* dacă relaţia este adevărată şi *false* în caz contrar.

Operatorii relaţionali = = şi != funcţionează de asemenea şi cu obiecte, însă rezultatele pe care le generează pot fi uneori confuze pentru un novice în Java. Exemplul care urmează:

```
public class echivalenta {
  public static void main(String[] arg) {
    int i=10;
    int j=10;
    System.out.println(i==j);
    System.out.println(i!=j);
    Integer a = new Integer(10);
    Integer b = new Integer(10);
    System.out.println(a==b);
    System.out.println(a!=b);
  }
}
```

produce următoarea ieşire:

```
true
false
false
true
```

Exemplul nr. 4 – Operatorii relaţionali în cazul obiectelor

La prima vedere poate părea anormal ca în cazul expresiei logice **a = = b** rezultatul returnat să fie *false*, însă dacă ţinem seama de faptul că **a** şi **b** sunt referinţe la obiecte (deci adrese) şi nu valori de tipul **int** ne dăm seama că rezultatul returnat este corect. Problema care se pune în mod natural este aceea de a găsi un mecanism prin care să comparăm efectiv conţinutul a două obiecte; pentru aceasta, în Java este implementată metoda **equals()** care există pentru toate clasele din biblioteca Java.

```
public class egalitate {
  public static void main(String[] arg) {
    Integer a = new Integer(10);
    Integer b = new Integer(10);
    System.out.println(a.equals(b));
  }
}
```

rezultatul este:

```
true
```

Exemplul nr. 5 – Folosirea metodei *equals* pentru compararea conţinutului obiectelor

Important este faptul că evaluarea expresiilor logice se face prin metoda **scurtcircuitării** care face ca evaluarea să se oprească în momentul în care valoarea de adevăr a expresiei este sigur determinată). Aceasta înseamnă că pot exista cazuri în care

programul să nu analizeze până la capăt expresia logică dacă pe parcursul evaluării ei se poate determina clar rezultatul final.

Astfel, dacă i=2, j=8 şi k=10, în cazul unei expresii de genul

if ((i<j) && (j<k))

sunt evaluate ambele componente (i<j) , (j<k), însă dacă i=2, j=1 şi k=10, este evaluată doar componenta (i<j) deoarece după evaluarea ei se poate spune cu precizie că întreaga expresie are valoarea de adevăr *false*, indiferent de valorile de adevăr ale celorlalte componente.

Pentru o mai bună înţelegere a conceptului de evaluare prin metoda scurtcircuitului, prezentăm următorul exemplu:

```
public class ScurtCircuit {
  static boolean eval_unu(int val) {
    System.out.println("eval_unu " + (val < 10) );
  return val<10;
  }
  static boolean eval_doi(int val) {
    System.out.println("eval_doi " + (val > 10) );
  return val>10;
  }
  public static void main(String[] arg) {
    if (eval_unu(15) && eval_doi(5))
      System.out.println("Expresia are valoarea true");
    else
      System.out.println("Expresia are valoarea false");
  }
}
```

care produce rezultatul:

```
eval_unu false
Expresia are valoarea false
```

Exemplul nr. 6 – Evaluarea prin scurtcircuitare

La prima vedere, rezultatul ar fi trebuit să fie următorul:

```
eval_unu false
eval_doi true
Expresia are valoarea false
```

deoarece în partea condiţională a instrucţiunii **if** ar trebui evaluate atât *eval_unu* cât şi *eval_doi*.

Atunci când se evaluează expresia *eval_unu*, se apelează de fapt metoda corespunzătoare care întoarce ca rezultat valoarea *false*. Având în vedere faptul că expresia din interiorul instrucţiunii **if** este un AND logic, este clar că, indiferent de valorile pe care le vor avea restul expresiilor componente, valoarea întregii expresii va fi *false* şi ca urmare nu mai are rost continuarea evaluării restului expresiei. În acest moment are loc ”scurtcircuitarea” procesului de evaluare.

- **operatorul** if-else: exp_logica ? val_pt_true : val_pt_false;

Acest operator este poate puţin mai neobişnuit deoarece are trei operanzi; el este în mod sigur un operator deoarece returnează întotdeauna o valoare, spre deosebire de structura clasică de control if-(then)-else care poate să nu returneze o valoare.

Dacă expresia logică este evaluată la valoarea *true*, atunci se returnează valoarea specificată prin *val_pt_true*, iar în caz contrar se returnează *val_pt_false*.

Operatorul condiţional din exemplul următor este echivalent cu folosirea structurii de control *if-else*:

```
  static int operator(int a) {
    return a<20 ? a*10 : a*5;
  }
```

```
  static int operator(int a) {
if (a<20)
        return a*10;
else
        return a*5;
  }
```

Exemplul nr. 7 – Operatorul relaţional condiţional

- **operatorul** + pentru concatenarea şirurilor

După cum deja s-a văzut, pentru concatenarea şirurilor se foloseşte operatorul **+**. Acest lucru este natural la prima vedere însă în spatele acestui mecanism atât de simplu pentru programator se află de fapt supraîncărcarea operatorului **+** despre care vom discuta mai detaliat cu altă ocazie. Dacă o expresie începe cu un String, atunci toţi operanzii care urmează trebuie să fie tot de tip String; dacă acest lucru nu se întâmplă, Java va încerca să realizeze conversia de tip astfel încât întreaga expresie să conţină doar String-uri. Astfel secvenţa:

```
String S = ”abcde”;
int val_1 = 200;
int val_2 = 300;
System.out.println(S + val);
```

va avea ca rezultat afişarea şirului abcde200300.

- **operatori de conversie (cast)**: (tip_data)

Aceşti operatori au rolul de a ”compatibiliza” tipurile de date în cazul unei operaţii sau a unei evaluări într-o expresie. Java

converteşte automat un tip de dată în altul atunci când este posibil, iar conversia se poate executa în două sensuri:

a) conversie prin contracţie
   int i = 50;
   long a = (long) i; *//conversie prin extensie*
b) conversie prin extensie
   long j = 30;
   int b = (int) j; *//conversie prin contracţie*

1.2. Structuri de control a execuţiei programelor Java

Java oferă programatorului o serie de structuri de control flexibile care permit un control perfect al execuţiei programului.

1.2.1. Structura de control alternativă if-else

Structura de control alternativă **if-else** are următoarea formă:

if (expresie_logică)
    bloc_instrucţiuni

sau

if (expresie_logică)
    bloc_instrucţiuni1
else
    bloc_instrucţiuni2

După cum se observă, partea de **else** este opţională; blocul de instrucţiuni poate fi format dintr-o singură instrucţiune urmată de semnul ”**;**” sau din mai multe instrucţiuni grupate între acolade. Pentru exemplul următor:

```
public class IfElse {
  public static void main(String[] arg) {
    System.out.println("Primul numar este " + test(10,20));
  }
  static String test(int a, int b) {
    if (a>b)
      return "mai mare";
    else
      return "mai mic";
  }
}
```

rezultatul afişat va fi:

```
Primul numar este mai mic
```

Exemplul nr. 8 – Structura if-else în Java

Cuvântul cheie **return** are două efecte: în primul rând precizează ce valoare va fi returnată de către metoda în cadrul căreia este folosită şi în al doilea rând determină returnarea imediată a acelei valori.

1.2.2. Structuri de control repetitive

În Java există mai multe structuri de control pentru reprezentarea iteraţiilor: **while**, **do-while**, **for**.

Atunci când se doreşte repetarea unei secvenţe atât timp cât este îndeplinită o anumită condiţie, se recomandă folosirea structurii **while** care are următoarea formă:

while (expresie_logică)
    bloc_de_instrucţiuni;

Expresia_logică este evaluată înainte de prima iteraţie şi apoi după fiecare iteraţie. În exemplul următor se afişează toate numerele pare cuprinse între 0 şi 100.

```
public class whileTest {
  public static void main(String[] arg) {
    int i = 0;
    while (i<=100) {
      System.out.println("S-a ajuns la numarul " + i);
      i = i + 2;
    }
  }
}
```

Exemplul nr. 9 – While în Java

Este posibil ca expresia logică evaluată să aibă valoarea false încă de la început şi astfel blocul de instrucţiuni să nu se execute niciodată.

În cazul în care se doreşte ca secvenţa de instrucţiuni să se execute cel puţin o dată şi apoi să se verifice la fiecare iteraţie îndeplinirea unei condiţii logice, se foloseşte structura **do-while** care are următoarea formă:

do
    secvenţă_instrucţiuni
while (expresie_logică**)**;

Structurile **while** şi **do-while** se folosesc atunci când nu se cunoaşte cu precizie de la început numărul de iteraţii.

Structura de control **for** oferă posibilitatea de a iniţializa una sau mai multe variabile şi de a le incrementa sau decrementa pe parcursul execuţiei secvenţei de instrucţiuni atât timp cât este îndeplinită o anumită condiţie logică. Forma generală este următoarea:

for(iniţializare; expresie_logică; pas_iteraţie)
    secvenţă_instrucţiuni

Oricare dintre componentele **for** (adică iniţializarea, expresia logică şi iteraţia) pot să lipsească. Expresia logică este testată înaintea fiecărei iteraţii şi execuţia secvenţei de instrucţiuni va continua atât timp cât ea are valoare true. În momentul în care expresia logică are valoarea false, controlul execuţiei este predat instrucţiunii imediat următoare secvenţei de instrucţiuni din cadrul **for**. La sfârşitul fiecărei iteraţii se execută pasul de iteraţie.

Concret, pentru a calcula şi afişa suma numerelor cuprinse între 1 şi 50 putem folosi următoarea secvenţă:

```
public class forTest {
 public static void main(String[] arg) {
  int suma = 0;
  for(int i=1;i<=50;i++)
    suma += i;    //echivalent cu suma=suma+i
  System.out.println("Suma primelor 50 de numere este " + suma);
  }
}
```

Exemplul nr. 10 – Utilizarea *for* în Java

În secţiunea **for** se pot declara varibilele care sunt folosite în iteraţie şi al căror domeniu de vizibilitate se limitează la blocul de instrucţiuni din interiorul buclei for.

De asemenea, se pot folosi mai multe variabile care vor fi folosite pentru realizarea iteraţiilor, conform modelului următor:

```
for (int i=1, j=5; i<10 && j<20; i++, j++)
        /*bloc de instrucţiuni*/
```

Pentru parcurgerea completă a unui vector se recomandă folosirea buclei **for** în maniera:

```
public class forTest2 {
 public static void main(String[] arg) {
  String[ ] Nume = {"Ionescu", "Popescu", "Vasilescu"};
  for(int i=0;i<Nume.length;i++)
      System.out.println(Nume[i]);
  }
}
```

Exemplul nr. 11 – "Parcurgerea unui vector folosind instrucţiunea *for*"

1.2.3. Structura alternativă generalizată

În Java, această strcutură este implementată prin cuvântul cheie ***switch*** care permite selecţia multiplă în maniera următoare:

```
switch (selector) {
        case valoare1: secvenţă1; break;
        case valoare2: secvenţă2; break;
        case valoare3: secvenţă3; break;
                // ...
        default: secvenţă;
}
```

Switch compară valoarea selectorului cu fiecare valoare specificată prin *case*; cuvântul cheie *break* determină determină predarea controlului execuţiei după corpul secvenţei *switch*.

```
public class switchTest {
  public static void main(String[] arg) {
   int nr_copii;   //numarul de copii aflati in intretinere
   float procent;  //procentul pentru deducerea personala suplimentara
   nr_copii = 2;
      switch (nr_copii) {
        case 0: procent = 0;break;
        case 1: procent = 10;break;
        case 2: procent = 15;break;
        case 3: procent = 18;break;
        default: procent = 20; //adica pentru mai mult de 3 copii
      }
  System.out.println("Deducerea personala suplimentara este: " + procent
+ "%");
  }
}
```

Exemplul nr. 12 – Folosirea structurii switch-case în Java

Aceasta este maniera clasică şi cea mai folosită pentru construirea unei structuri alternative generalizate în Java, însă prezenţa lui *break* este opţională. Dacă lipseşte *break*, se execută următoarele secvenţe case până în momentul în care este întâlnit primul *break*.

Ultima secvenţă din secţiunea *default* nu are nevoie de *break* deoarece după execuţia ei se iese din structura *switch*.

Structura *switch-case* este foarte utilă atunci când selectorul este de tip int sau char, însă ea nu funcţionează în cazul în care selectorul este de alt tip.

EXEMPLE_LABORATOR:

LABORATOR 1

Să se realizeze o secvenţă care să calculeze salariile lunare şi salariul anual total al unei persoane în funcţie de numărul de zile lucrate în fiecare lună.

Salariul lunar = număr de zile lucrate * salariul zilnic

Salariul anual total = $\sum salariul_luna$

```
public class Persoana_simpla {
   public static void main(String[] argument) {
      String nume="Popescu";
      String prenume="Vasile";
        int[] zile_lucrate = new int[12];  //numarul de zile lucrate in fiecare
luna a anului
      final float salariu_zilnic;
      float salariu_total, salariu_lunar;
      zile_lucrate[0]=20; //ianuarie
      zile_lucrate[1]=21;
      zile_lucrate[2]=23;
      zile_lucrate[3]=22;
      zile_lucrate[4]=24;
      zile_lucrate[5]=17;
      zile_lucrate[6]=21;
      zile_lucrate[7]=20;
      zile_lucrate[8]=20;
      zile_lucrate[9]=22;
      zile_lucrate[10]=21;
      zile_lucrate[11]=19;  //decembrie
      salariu_zilnic = 110000;  //stabilirea salariului zilnic
      //calculul salariilor lunare si a salariului anual total
      salariu_total = 0;
      for(int i=0;i<=11;i++) {
         salariu_lunar = salariu_zilnic * zile_lucrate[i];
             System.out.println("Salariul pe luna " + (i+1) + " este de " +
(int)salariu_lunar);
         salariu_total = salariu_total + salariu_lunar;
      }
          System.out.println("Salariul anual total pentru " + nume +" " +
prenume + " este de " + (int)salariu_total);
      //pe linia de mai sus se realizeaza conversia prin contractie
      //pentru a putea sa se afiseze valori inteligibile
   }
   public Persoana_simpla() {
   }
}
```

Rezultatul care se obţine:

```
Salariul pe luna 1 este de 2200000
Salariul pe luna 2 este de 2310000
Salariul pe luna 3 este de 2530000
Salariul pe luna 4 este de 2420000
Salariul pe luna 5 este de 2640000
Salariul pe luna 6 este de 1870000
Salariul pe luna 7 este de 2310000
Salariul pe luna 8 este de 2200000
Salariul pe luna 9 este de 2200000
Salariul pe luna 10 este de 2420000
Salariul pe luna 11 este de 2310000
Salariul pe luna 12 este de 2090000
Salariul anual total pentru Popescu Vasile este de 27500000
```

TEMA1 – de rezolvat în laborator:

Salariul zilnic diferă în funcţie de fiecare lună şi ca urmare trebuie implementată o secvenţă care să definească un nou vector pentru salariul_zilnic_lunar şi care să calculeze salariile lunare în funcţie de cei doi vectori: **zile_lucrate[ ]** şi **salariul_zilnic_lunar[ ]**.

```
public class Persoana_simpla_tema1 {
    public static void main(String[] argument) {
        String nume="Popescu";
        String prenume="Vasile";
          int[] zile_lucrate = new int[12];  //numarul de zile lucrate in fiecare
luna a anului
        float[] salariu_zilnic_lunar = {110000,100000,105000,120000,
                                 98000,112000,115000,110000,
                                         100000,100000,103000,100000}; //o alta
posibilitate de a initializa vectorii
        float salariu_zilnic, salariu_total, salariu_lunar;
        zile_lucrate[0]=20; //ianuarie
        zile_lucrate[1]=21;
        zile_lucrate[2]=23;
        zile_lucrate[3]=22;
        zile_lucrate[4]=24;
        zile_lucrate[5]=17;
        zile_lucrate[6]=21;
        zile_lucrate[7]=20;
        zile_lucrate[8]=20;
        zile_lucrate[9]=22;
        zile_lucrate[10]=21;
        zile_lucrate[11]=19;  //decembrie
        //calculul salariilor lunare si a salariului anual total
        salariu_total = 0;
        for(int i=0;i<=11;i++) {
            salariu_lunar = salariu_zilnic_lunar[i] * zile_lucrate[i];
                System.out.println("Salariul pe luna " + (i+1) + " este de " +
(int)salariu_lunar);
            salariu_total = salariu_total + salariu_lunar;
```

```
        }
            System.out.println("Salariul anual total pentru " + nume +" " +
prenume + " este de " + (int)salariu_total);
        //protected linia de mai sus se realizeaza conversia prin contractie
        //pentru a putea sa se afiseze valori inteligibile
    }
```

TEMA2 – de rezolvat în laborator

Modificati codul de mai sus astfel încât numele şi prenumele să fie stabilite la instanţierea unei persoane, iar stabilirea **zile_lucrate[ ]** şi **salariu_zilnic_lunar[ ]** să se realizeze prin apelarea unor metode corespunzătoare. Calculul salariilor lunare şi a celui total să se facă prin metoda **Calcul_salarii( )**..

Clasa CPersoana (cu atributele şi metodele specifice)

```
public class CPersoana {
   //membrii clasei
   //atributele
   private String nume;
   private String prenume;
   private int[] zile_lucrate = new int[12];
   private float[] salariu_zilnic_lunar = {0,0,0,0,0,0,0,0,0,0,0,0};
   private float salariu_lunar=0;
   private float salariu_total=0;

   //metodele
   //constructorul
   public CPersoana(String Num, String Pren) {
    nume = Num;
    prenume = Pren;
   }

   //setarea zilelor lucrate
   public void setZileLucrate(int[] zile) {
        //realizez copierea parametrului zile in membrul private zile_lucrate
       System.arraycopy(zile, 0, zile_lucrate, 0, 12);
   }

   //setarea salariului zilnic pentru fiecare luna
   public void setSalariuZilnic(float[] sal) {
                 //realizez copierea parametrului sal in membrul private
salariu_zilnic_lunar
       System.arraycopy(sal,0, salariu_zilnic_lunar, 0, 12);
   }

   //afisarea salariilor lunare si a salariului total
    public void afiseaza_salarii() {
```

```
        //calculul salariilor lunare si a salariului anual total
        salariu_total = 0;
        for(int i=0;i<=11;i++) {
            salariu_lunar = salariu_zilnic_lunar[i] * zile_lucrate[i];
                System.out.println("Salariul pe luna " + (i+1) + " este de " +
(int)salariu_lunar);
            salariu_total = salariu_total + salariu_lunar;
        }
            System.out.println("Salariul anual total pentru " + nume +" " +
prenume + " este de " + (int)salariu_total);
        //protected linia de mai sus se realizeaza conversia prin contractie
        //pentru a putea sa se afiseze valori inteligibile
    }
}
```

Clasa LPersoane din care se apelează instanţele şi metodele clasei CPersoana.

```
public class LPersoane {
    public static void main(String[] arg) {
        CPersoana pers1 = new CPersoana("Ionescu","Ion");
        int[] zile = {20,21,20,19,22,22,21,20,11,19,20,22};
                                                        float[]      sal_zilnic      =
{100000,110000,105000,120000,100000,130000,99000,95000,110000,1
05000,100000,99000};
        pers1.setZileLucrate(zile);
        pers1.setSalariuZilnic(sal_zilnic);
        pers1.afiseaza_salarii();
    }
    public LPersoane() {
    }
}
```